# An Enhanced Logical Benders Approach for Linear Programs with Complementarity Constraints

Francisco Jara-Moroni  John E. Mitchell  Jong-Shi Pang
Andreas Wächter

2019-03-11

**Abstract**

This work extends the ones of Hu et al. Hu et al. (2008) and Bai et al. Bai et al. (2013) of a logical Benders approach for globally solving Linear Programs with Complementarity Constraints. By interpreting the logical Benders method as a reversed branch-and-bound method, where the whole exploration procedure starts from the leaf nodes in an enumeration tree, we provide a new framework over which we can combine master problem and cut generation in a single process. We also present an optimization-based sparsification process which makes the cut generation more efficient. Numerical results are presented to show the effectiveness of this unified method. Results are also extended to larger complementarity dimensions, exceeding what the original method has been able to handle.

**Keywords:** complementarity constraints, branch-and-bound, global optimization, logical Benders

## 1 Introduction

This paper focuses on the global solution of Linear Programs with Complementarity Constraints (LPCCs) in its general form

$$\begin{array}{llll} \underset{x,\,y,\,w}{\text{minimize}} & & g^T x & \\ \text{subject to} & & A_I x + B_I y + C_I w & \leq b_I \\ & & A_E x + B_E y + C_E w & = b_E \\ & 0 \leq & y \perp w & \geq 0 \end{array} \tag{1}$$

where $x \in \mathbb{R}^{n_x}$, $y, w \in \mathbb{R}^{n_c}$, $b_I \in \mathbb{R}^{k_I}$ and $b_E \in \mathbb{R}^{k_E}$. Dimensions for the matrices $A_I$, $B_I$, $C_I$, $A_E$, $B_E$ and $C_E$ are defined accordingly. Notice that the

objective function only depends on the variable $x$. This is not restrictive in the formulation since any dependence on $y$ and/or $w$ can be taken into account within the equality constraints.

The LPCC has grown in importance in applications during the last forty years, beginning with the early work in the integer programming community on what was known then as the "complementary program" (Ibaraki, 1971, 1973; Jeroslow, 1978), and as a special case of a mathematical program with equilibrium constraints (MPECs) (Luo et al., 1996) on which there is an extensive literature to date. The reference (Hu et al., 2012b) gathers the different applications of LPCCs arising from science and engineering, as complementarity constraints can be used to model many logical, piecewise, and nonconvex conditions (Hu et al., 2012b), even discontinuous ones such as cardinality objective (Feng et al., 2018) and constraints (Burdakov et al., 2016). In addition, the LPCC provides an interesting framework for the study of nonconvex quadratic programs (Hu et al., 2012a). In general, determining the global solution of LPCCs is NP-hard. By global solution of the LPCC we mean certifying the problem is in one of its three possible states: infeasible, unbounded below or having a finite optimal solution. If both $y$ and $w$ are bounded, then it is well known that (1) can be reformulated as a Mixed Integer Linear Program (MILP), by setting diagonal matrices $M_y$ and $M_w$ and a binary vector $p \in \{0,1\}^{n_c}$ representing the complementarity between the variables $y$ and $w$;

$$\begin{array}{lll} \underset{x,\,y,\,w,\,p}{\text{minimize}} & g^T x & \\ \text{subject to} & A_I x + B_I y + C_I w & \leq b_I \\ & A_E x + B_E y + C_E w & = b_E \\ & w & \leq M_w p \\ & y & \leq M_y (1-p) \\ & y, w & \geq 0. \end{array} \tag{2}$$

The main challenge with this formulation is the computation of valid bounds to obtain the diagonal values of matrices $M_y$ and $M_w$, if these bounds are not explicitly available. Furthermore, if either $y$ or $w$ is not bounded then this reformulation cannot be applied.

Branch-and-bound approaches based on the big-M formulation (2) often run into numerical difficulties for two principal reasons: (i) the LP relaxations are weak, with small fractional values of the variables $p$ allowing both sides of the complementarity to be violated, and (ii) if the integrality tolerance is not tight enough then a solution that is returned as integral may violate both sides of the complementarity Belotti et al. (2016); Bonami et al. (2015). Recently, there has been interest in using indicator constraints to represent disjunctions, and these are available in commercial packages including CPLEX Cplex (2010) and GuRoBi Optimization (2014). For example, the complementarity constraint $0 \leq y \perp w \geq 0$ in the scalar variables $y$ and $w$ could be represented using

indicator constraints as

$$p = 0 \implies w = 0$$
$$p = 1 \implies y = 0$$

with $p$ a scalar binary variable. Indicator constraints may be more numerically robust than big-M formulations, but they may lead to weaker LP relaxations. For more detailed discussion of indicator constraints and the difficulties with big-$M$ formulations, see, for example, Belotti et al. (2016); Bonami et al. (2015). The paper Yu et al. (2018) developed a branch-and-bound algorithm for LPCCs and compared it with using indicator constraint and big-$M$ formulations in CPLEX for various classes of LPCCs. The implementation in (Yu et al., 2018) used preprocessing to tighten bounds on variables and add other cutting planes, with the same preprocessed problems used in all branching algorithms. The specialized branching rules developed in (Yu et al., 2018) led to strong computational results, significantly outperforming the native CPLEX approaches with either indicator constraints or big-$M$ formulations. Thus, it appears there is a need to develop better branching rules for handling complementarity constraints (see also the computational results in (Belotti et al., 2016; Fischer and Pfetsch, 2018)). Logical Benders decomposition gives an alternative method to generate branching rules for these problems. In the current paper, we improve the state of the art in logical Benders decomposition, bringing the computational performance closer to that in (Yu et al., 2018).

While there have been significant advances on computational methods based on non-linear programming (NLP) to solve MPCCs, their main focus is to find some type of stationary point in an efficient manner. Solvers such as FILTER (Fletcher* and Leyffer, 2004) and KNITRO Leyffer et al. (2006), based on sequential quadratic programming and interior point methods, respectively, are capable of finding solutions very quickly, but have no guarantees on the quality of the computed solution. Since an LPCC can be interpreted as a disjunctive linear optimization problem, global solution methods are mainly based on some form of enumeration schemes. Therefore, many integer programming based approaches have been tested on LPCC, combining essentially branch-and-bound and cutting plane methods (see (Bard and Moore, 1990; Jeroslow, 1978; Ibaraki, 1971, 1973; Yu, 2011; Yu et al., 2018)).

Based on the works of Hooker and Ottosson (Hooker and Ottosson, 2003), a logical Benders approach was developed for LPCCs (Hu et al., 2008) and later extended to Q(uadratic)PCCs (Bai et al., 2013), where the complementarity pieces are described by binary variables, and later discarded for exploration via a logical Benders cut generation scheme. Although originally stated as an extension to these logical Benders based approaches, the method presented in this paper is also closely related to branch-and-bound as it will be described in the next section.

A first intention of this paper is to establish a relationship between logical Benders and branch-and-bound methods and, taking advantage of this, we develop a more efficient and robust way of selecting and discarding pieces within the logical Benders framework. We also complement the Benders' cut generation scheme with an optimization-based procedure in order to strengthen these cuts.

The paper is organized as follows. Section 1 introduces the baseline Benders method in detail. We describe the relationship between Benders and branch-and-bound in this context in Section 2. In Section 3 the enhancements to logical Benders, from a branch-and-bound perspective, are described. Finally, numerical results supporting the effectiveness of these enhancements are presented in Section 4.

### 1.1 Problem statement

For the remainder of this section, the following set is defined $\Omega_P := \{(x, y, w) | A_I x + B_I y + C_I w \leq b_I;\ A_E x + B_E y + C_E w = b_E;\ y, w \geq 0\}$, which represents the feasible region of (1) without the complementarity constraint.

The general idea of the algorithm presented in this paper follows the one espoused by Hu et al. (2008). A master problem selects different complementarity pieces which are solved to optimality and the corresponding solution is stored as the incumbent if it is the best one found so far. Pieces are discarded by means of a cut generation method in the master problem. The method ends when all pieces have been explored or discarded. In the following subsection we describe these computational steps in more detail.

### 1.2 Logical Benders Decomposition

Let $\mathcal{N} := \{1, \ldots, n_c\}$. Given a pair of disjoint subsets $(I^w, I^y)$ of $\mathcal{N}$, the relaxed subproblem for (1) defined by $I^w$ and $I^y$ is the minimization of $g^T x$ over $\Omega_P$ with the added restrictions $w_i \leq 0$, $i \in I^w$ and $y_i \leq 0$, $i \in I^y$. That is,

$$\begin{aligned} \phi_P(I^w, I^y) = \quad & \underset{(x,\,y,\,w)\in\Omega_P}{\text{minimize}} \quad g^T x \\ & \text{subject to} \quad w_i \quad \leq 0 \text{ for } i \in I^w \quad (\lambda_i^w) \\ & \qquad\qquad\quad y_j \quad \leq 0 \text{ for } j \in I^y \quad (\lambda_j^y). \end{aligned} \tag{3}$$

where the $\lambda$'s in parentheses represent the respective dual variables. In the case that $(I^w, I^y)$ is a partition of $\mathcal{N}$, we have $w^T y = 0$ for every feasible point of (3), and we refer to its feasible region as a complementarity piece. In our setting, these partitions will be described by a binary vector $p \in \{0, 1\}^{n_c}$ in the following way:

$$\begin{aligned} \phi_P(p) = \quad & \underset{(x,\,y,\,w)\in\Omega_P}{\text{minimize}} \quad g^T x \\ & \text{subject to} \quad w_i \quad \leq 0,\ i : p_i = 0 \quad (\lambda_i^w) \\ & \qquad\qquad\quad y_j \quad \leq 0,\ j : p_j = 1 \quad (\lambda_j^y) \end{aligned} \tag{4}$$

In this case $I^w = \bar{I}^w(p) := \{i : p_i = 0\}$ and $I^y = \bar{I}^y(p) := \{i : p_i = 1\}$. It is clear that for any vector $(x, y, w)$ feasible in (1) there exists a binary vector $p$ such that $(x, y, w)$ is also feasible in (4). By convention we assume that $\phi_P(p) = \infty$ if (4) is infeasible. Therefore, (1) is equivalent to

$$\underset{p\in\{0,1\}^{n_c}}{\text{minimize}} \quad \phi_P(p). \tag{5}$$

Instead of exploring all $2^{n_c}$ possible pieces explicitly, we maintain a master problem that keeps track of all the pieces that still need to be explored. We denote the state of the master problem by a set $\mathcal{C}$ that consists of pairs $(I^w, I^y)$ of disjoint subsets $I^w$ and $I^y$ of $\mathcal{N}$. Then the set of pieces $p$ that still have to be explored is given by

$$\mathcal{F} = \left\{ p \in \{0,1\}^{n_c} : \sum_{i \in I^w} p_i + \sum_{i \in I^y} (1 - p_i) \geq 1 \text{ for all } (I^w, I^y) \in \mathcal{C} \right\}. \tag{6}$$

We refer to an inequality in the above definition as a "cut" in the master problem, and use the corresponding sets $I^w$ and $I^y$ to denote the cut. For example, once (4) has been solved for a given piece $p$, we could add the cut defined by $I^w = \bar{I}^w(p)$ and $I^y = \bar{I}^y(p)$ to $\mathcal{C}$. In this way, the piece $p$ is no longer in the set of unexplored pieces $\mathcal{F}$. If $I^w \subseteq \bar{I}^w(p)$ and $I^y \subseteq \bar{I}^y(p)$ is not a partition of $\mathcal{N}$, also other pieces will be removed by the corresponding cut. Clearly, the fewer components are in $I^w$ and $I^y$, the fewer the pieces left in $\mathcal{F}$. The processes in which these components are removed from $I^w$ or $I^y$ will be referred to as "sparsification", and are discussed in detail in Section 3.2.

### 1.2.1 Algorithm Outline

In each iteration of the logical Benders algorithm, a pair $(\tilde{I}^w, \tilde{I}^y)$ is chosen such that

$$\tilde{I}^w \cap I^y \neq \emptyset \text{ or } \tilde{I}^y \cap I^w \neq \emptyset, \text{ for all pairs } (I^w, I^y) \in \mathcal{C}. \tag{7}$$

Then the subproblem (3) corresponding to $(\tilde{I}^w, \tilde{I}^y)$ is solved, and based on the outcome a new cut $(I^w, I^y)$ is added to $\mathcal{C}$. If $(\tilde{I}^w, \tilde{I}^y)$ is a partition described by a binary vector $\tilde{p}$, condition (7) is equivalent to $\tilde{p}$ being in $\mathcal{F}$. The new cut must at least remove $\tilde{p}$ from $\mathcal{F}$. Also, if (4) is feasible for $\tilde{p}$, we obtain a feasible point for the original problem. The algorithm keeps the best feasible point encountered so far as the incumbent, together with its optimal objective value $U$, which is an upper bound for the optimal objective value of (1).

Since there are only finitely many pieces, $\mathcal{F}$ must eventually become empty. All pieces have been (implicitly) explored and the algorithm terminates. The current incumbent is an optimal solution for (1). If no incumbent has been found, the original problem is infeasible. And if during any iteration the algorithm finds a piece such that (4) is unbounded below, then (1) is unbounded.

### 1.2.2 Cut generation

For a given piece $p$, any pair $(I^w, I^y)$ such that $I^w \subseteq \bar{I}^w(p)$ and $I^y \subseteq \bar{I}^y(p)$ makes subproblem (3) a relaxation of (4), and therefore we have $\phi_P(I^w, I^y) \leq \phi_P(p)$. Again, we define $\phi_P(I^w, I^y) = \infty$ if (3) is infeasible. Note that (3) is also a relaxation of (4) for any other $\hat{p}$ such that $I^w \subseteq \bar{I}^w(\hat{p})$ and $I^y \subseteq \bar{I}^y(\hat{p})$, and therefore $\phi_P(I^w, I^y) \leq \phi_P(\hat{p})$.

Now suppose that $U$ is the objective value for the current incumbent and therefore an upper bound on the optimal objective function. In our search for

**Algorithm 1** Logical Benders: baseline method

1: Let $\mathcal{C} = \emptyset$, $U = \infty$.
2: **while** $\mathcal{F} \neq \emptyset$ **do**
3: Select a pair $(\bar{I}^w, \bar{I}^y)$ satisfying (7).
4: Evaluate $\phi_P(\bar{I}^w, \bar{I}^y)$.
5: Update $U$ and store incumbent if any.
6: Find $(I^w, I^y)$ such that $\bar{I}^w \cap I^y = \emptyset$, $\bar{I}^y \cap I^w = \emptyset$ and $\phi_P(I^w, I^y) \geq U$.
7: Add cut $(I^w, I^y)$ to $\mathcal{C}$.
8: Return incumbent as optimal solution or output infeasible or unbounded.

a better incumbent (if possible), we need to find a piece $\hat{p}$ so that $\phi_P(\hat{p}) < U$. Consequently, if $\phi_P(I^w, I^y) \geq U$, all pieces $\hat{p}$ with $I^w \subseteq \bar{I}^w(\hat{p})$ and $I^y \subseteq \bar{I}^y(\hat{p})$ cannot contain a better incumbent, and we can exclude them all from $\mathcal{F}$. This is done by adding the cut $(I^w, I^y)$ to $\mathcal{C}$. This baseline method in its most general form is described in Algorithm 1.

In the setting of Hu et al. (2008), step 3 always selects a partition of $\mathcal{N}$, described by a vector $\bar{p}$, satisfying (7) which is equivalent to $\bar{p} \in \mathcal{F}$. Steps 4 and 5 translate into comparing $\phi_P(p)$ with $U$, where an update is performed every time that $\phi_P(p) < U$. In order to find suitable subsets $I^w$ and $I^y$ in step 6, we consider the dual to (4):

$$
\begin{aligned}
\phi_D(I^w, I^y) = \quad \underset{\mu_I,\, \mu_E,\, \lambda^w,\, \lambda^y}{\text{maximize}} \quad & -b_I^T \mu_I + b_E^T \mu_E \\
\text{subject to} \quad & -A_I^T \mu_I + A_E^T \mu_E && = g \\
& -B_I^T \mu_I + B_E^T \mu_E - \lambda^y && \leq 0 \\
& -C_I^T \mu_I + C_E^T \mu_E - \lambda^w && \leq 0 \\
& \textstyle\sum_{i \notin I^w} \lambda_i^w + \sum_{i \notin I^y} \lambda_i^y && = 0 \\
& \mu_I,\, \lambda^w,\, \lambda^y && \geq 0,
\end{aligned} \tag{8}
$$

In (3), there are constraints on $w_i$ for $i \in I^w$, and the dual should contain only the corresponding multipliers $\lambda_i^w$. To simplify the notation, the multipliers for $w \leq 0$ have been extended to a full vector $\lambda^w \in \mathbb{R}^{n_c}$, and $\lambda^y$ is similarly defined. The constraint $\sum_{i \notin I^w} \lambda_i^w + \sum_{i \notin I^y} \lambda_i^y = 0$ makes sure that the newly introduced components of $\lambda^w$ and $\lambda^y$ must be zero, so that (8) is indeed the dual of (3).

Assuming (4) has a finite optimal solution, we have $\phi_P(I^w, I^y) = \phi_D(I^w, I^y)$. It is then clear that $(I^w, I^y)$ defines a valid cut if $\phi_D(I^w, I^y) \geq U$, or, equiva-

lently, the set

$$\Omega_D(I^w, I^y) := \left\{ (\mu_I,\, \mu_E,\, \lambda^w,\, \lambda^y) : \begin{array}{rl} -b_I^T\mu_I + b_E^T\mu_E & \geq U \\ A_I^T\mu_I + A_E^T\mu_E & = g \\ -B_I^T\mu_I + B_E^T\mu_E - \lambda^y & \leq 0 \\ -C_I^T\mu_I + C_E^T\mu_E - \lambda^w & \leq 0 \\ \sum_{i\notin I^w}\lambda_i^w + \sum_{i\notin I^y}\lambda_i^y & = 0 \\ \mu_I,\, \lambda^w,\, \lambda^y & \geq 0 \end{array} \right\} \quad (9)$$

is not empty.

The following procedure was suggested in (Bai et al., 2013) to obtain a valid cut: Given a piece $p$, solve the primal (4). If it is feasible, let $(\mu_I,\, \mu_E,\, \lambda^w,\, \lambda^y)$ be a dual optimal solution (with $\lambda^w$ and $\lambda^y$ properly extended). Now define $I^w = \{i \in \bar{I}^w(p) : \lambda_i^w > 0\}$ and $I^y = \{i \in \bar{I}^y(p) : \lambda_i^y > 0\}$. Then it is easy to see that $(\mu_I,\, \mu_E,\, \lambda^w,\, \lambda^y) \in \Omega_D(I^w, I^y)$ and therefore $(I^w, I^y)$ defines a valid cut. If the optimal solution in (4) is not strictly complementary, then this procedure produces a cut that removes more than just $p$ from $\mathcal{F}$. Note that this implies that the relaxation $\phi_P(I^w, I^y) \geq U$.

The reference Bai et al. (2013) discusses a procedure to sparsify the cut further. In their paper, the authors sort the sets $\{\lambda_i^w : i \in I^w\}$ and $\{\lambda_i^y : i \in I^y\}$ in descending order and define $\tilde{I}^w$ and $\tilde{I}^y$ as the top half of each set, respectively. They then solve the relaxed primal (3) to get $\phi_P(\tilde{I}^w, \tilde{I}^y)$. If $\phi_P(\tilde{I}^w, \tilde{I}^y) \geq U$, they set $(I^w, I^y) \leftarrow (\tilde{I}^w, \tilde{I}^y)$, which is also a valid cut. This process is repeated until $\phi_P(\tilde{I}^w, \tilde{I}^y) < U$.

A similar procedure is used if the primal problem is infeasible. In this case, the dual must have an unbounded ray that can be found by solving the homogeneous version of (8):

$$\begin{array}{rlrl} \phi_{D_0}(I^w, I^y) = & \underset{\mu_I,\, \mu_E,\, \lambda^w,\, \lambda^y}{\text{maximize}} & -b_I^T\mu_I + b_E^T\mu_E & \\ & \text{subject to} & -A_I^T\mu_I + A_E^T\mu_E & = 0 \\ & & -B_I^T\mu_I + B_E^T\mu_E - \lambda^y & \leq 0 \\ & & -C_I^T\mu_I + C_E^T\mu_E - \lambda^w & \leq 0 \\ & & \sum_{i\notin I^w}\lambda_i^w + \sum_{i\notin I^y}\lambda_i^y & = 0 \\ & & \mu_I,\, \lambda^w,\, \lambda^y & \geq 0. \end{array} \quad (10)$$

The set $\Omega_D$ with $g = 0$ and the constraint $-b_I^T\mu_I + b_E^T\mu_E \geq U$ replaced by $-b_I^T\mu_I + b_E^T\mu_E = 1$, will be denoted $\Omega_{D_0}$. As before, the generated cut $(I^w, I^y)$ guarantees that any piece removed by it will also be infeasible.

If the constraints of $\phi_P(I^w, I^y)$ (resp. $\phi_D(I^w, I^y)$) define an infeasible set then it will be understood that $\phi_P(I^w, I^y) = \infty$ (resp. $\phi_D(I^w, I^y) = -\infty$).

If at any iteration the algorithm finds a primal piece which is unbounded then (1) is also unbounded, so there is no more exploration required.

Initially the master problem feasible set $\mathcal{F}$ is $\{0,1\}^{n_c}$ and the upper bound $U = \infty$. A candidate complementarity piece $p$ is selected (by solving a satisfiability problem). If $\phi_{D_0}(p) = \infty$, a suitable cut is obtained from (10). Otherwise

if $\phi_P(p) = -\infty$ then the master problem is unbounded and the method stops. If $\phi_P(p) < U$ then the bound is updated ($U = \phi_P(p)$). Finally, a cut is obtained from (8). The obtained cut is then sparsified and added to the set $\mathcal{C}$. The process continues until $\mathcal{F}$ becomes infeasible. At that moment the algorithm returns the current incumbent.

### 1.3 Contributions of this work

There are two key drivers of the performance of this method: The selection of the pair $(I^w, I^y)$ and the strength (sparsity) of the generated cuts. Notice that both steps are linked, the generated cut depends on the selected subsets, and this selection depends on cuts that have already been generated.

In the methods that have been proposed in the past (Hu et al., 2008; Bai et al., 2013), the piece selection and the sparsification procedures were independent of each other. In fact, the candidate piece was chosen just as any $p \in \mathcal{F}$, without giving any preferences of one over another.

The contribution of this paper is that we consider the logical Benders algorithm from a different point of view, namely as a procedure that operates on a branch-and-bound tree (in reverse order compared to regular branch-and-bound methods). This relationship provides a justification for the selection of pairs $(I^w, I^y)$ and the subsequent sparsification procedure proposed in this paper. We show in Section 3 how a judicious selection of pairs, by exploiting this relationship, saves the logical Benders method a considerable amount of time and iterations.

## 2 Interpretation within Branch-and-Bound Framework

One well-known approach for solving MPCCs is based on branch-and-bound (B&B), introduced by Bard and Moore (1990). Efficient implementations of this framework include many enhancements, such as cutting planes and pseudocost branching (Yu, 2011; Yu et al., 2018). Here, we describe the basic B&B method with the purpose of interpreting the logical Benders algorithm as one that operates on a B&B tree. This will allow us to derive new piece selection and cut sparsification methods. The B&B method for LPCCs solves a collection of subproblems of the form (3), where $I^w$ and $I^y$ are disjoints subsets of $\mathcal{N}$. These subproblems correspond to nodes, denoted as $[I^w, I^y]$, in a binary enumeration tree (In our notation, we distinguish between cuts $(I^w, I^y)$ and nodes $[I^w, I^y]$ by their parentheses). The root node of such a tree corresponds to the subproblem described by $I^w = I^y = \emptyset$, i.e., none of the components of $y$ and $w$ are required to be complementary. Each node $[I^w, I^y]$ with $I^w \cup I^y \neq \mathcal{N}$ has two children. They are obtained by choosing a complementarity $j \in \mathcal{N} \setminus (I^w \cup I^y)$ that has not yet been fixed and by making $[I^w \cup \{j\}, I^y]$ and $[I^w, I^y \cup \{j\}]$ the index sets defining the children. We call $j$ the branching complementarity. Different trees are obtained by choosing different branching complementarities

at the nodes. Overall, there are $\Pi_{i=0}^{n_c}(n_c - i)^{2^i}$ possible trees $T$ corresponding to (1). (Although, in the regular B&B method, the full tree never needs to be completely constructed, since subtrees are eliminated from the search once it is clear that they cannot contain the optimal solution). The nodes on the last level with $I^w \cup I^y = \mathcal{N}$ are called leaves. Here, all complementarity constraints have been fixed at one of the two sides. If a leaf is feasible, its optimal solutions are feasible for the original problem (1). Therefore, each leaf corresponds to a piece $p$ in (1) and vice versa, with $I^w = \bar{I}^w(p)$ and $I^y = \bar{I}^y(p)$.

Since a child node $[\tilde{I}^w, \tilde{I}^y]$ is obtained by fixing a complementarity constraint to one of the two sides, its feasible region cannot be larger than that of its parent $[I^w, I^y]$. As a consequence, $\phi_P(\tilde{I}^w, \tilde{I}^y) \geq \phi_P(I^w, I^y)$. This includes the case in which subproblems are infeasible and a quantity in that relationship is $\infty$.

Given a node $[I_a^w, I_a^y]$ and one of its descendants $[I_d^w, I_d^y]$ in a particular tree, we denote the path $P$ from $[I_d^w, I_d^y]$ to $[I_a^w, I_a^y]$ by $j_1 \leftarrow j_2 \leftarrow j_3 \leftarrow \ldots \leftarrow j_L$, where $j_l$ are the branching complementarities that were added to obtain a child node from its parent. The indices are listed in order from the descendant to the ancestor; for example, $j_L$ represents the branching from node $[I_a^w, I_a^y]$.

## 2.1 Branch-and-Bound Algorithm

The B&B method constructs a tree during the course of the algorithm. It maintains a list of open nodes that have yet to be explored. At the beginning, this list is initialized with the root node. The root node corresponds to the subproblem described by $I^w = I^y = \emptyset$, i.e., none of the components of $y$ and $w$ are required to be complementary. Its optimal value $\phi_P(\emptyset, \emptyset)$ provides a lower bound on the optimal objective of the original problem (1). The method also stores an incumbent, which is a point that is feasible for the original problem (1) with the lowest objective value, call it $U$, found so far. Clearly, $U$ is an upper bound for the optimal objective value of (1). At the beginning, no incumbent is available, and we set $U \leftarrow \infty$.

In each iteration of the rudimentary B&B algorithm, an open node $[I^w, I^y]$ is chosen and the corresponding subproblem (3) is solved. There are four possible outcomes that determine the next step of the algorithm:

1. If (3) is feasible and $\phi_P(I^w, I^y) \geq U$, all descendants $[I_d^w, I_d^y]$ of this node must also have $\phi_P(I_d^w, I_d^y) \geq U$. Consequently, no feasible point for (1) can be found among the descendants of $[I^w, I^y]$ with a better objective value than $U$.

2. If $[I^w, I^y]$ is infeasible, also all its descendants must be infeasible, and again no better incumbent for (1) can be found below $[I^w, I^y]$.

3. If (3) is feasible and its computed optimal solution is feasible for the original problem (1) and $\phi_P(I^w, I^y) < U$, then the optimal solution of the subproblem, if it is finite, provides a new incumbent. The incumbent and the corresponding upper bound $U$ are updated.

4. If (3) is feasible but its computed optimal solution is not feasible for the original problem (1), then the set of complementarities that has not been fixed, $\mathcal{N} \setminus (I^w \cup I^y)$, must be non-empty. The algorithm then chooses a branching complementarity $j \in \mathcal{N} \setminus (I^w \cup I^y)$ and adds the corresponding children $[I^w \cup \{j\}, I^y]$ and $[I^w, I^y \cup \{j\}]$ to the list of open nodes.

In the cases 1–3, there is no need to explore descendants of the current node, since no feasible solution better than the incumbent can be found in that part of the tree. In that case, we call the current node a fathomed node. The B&B algorithm does not explicitly construct the part of the tree below a fathomed node.

The algorithm terminates once the list of open nodes becomes empty. The current incumbent is the optimal solution of (1). If no incumbent was found during the search, the original problem is infeasible. Since there is only a finite number of possible open nodes and no node can become an open node twice, the algorithm is guaranteed to terminate in a finite number of iterations.

## 2.2 Proving Optimality

For simplicity we will assume in the following that the optimal point is available as incumbent and that the purpose of the algorithm is to confirm its optimality. This is reasonable in a practical setting if we first solve a big-M formulation of (1), namely

$$\begin{aligned} \underset{(x,\,y,\,w)\in\Omega_P,\,p\in\{0,1\}^{n_c}}{\text{minimize}} \quad & g^T x \\ \text{subject to} \quad & w_j \le M p_j \text{ for } j \in \mathcal{N} \\ & y_j \le M(1-p_j) \text{ for } j \in \mathcal{N}, \end{aligned} \tag{11}$$

for a finite $M > 0$ with an MILP algorithm. In this manner, we can use powerful off-the-shelf MILP solver implementations. Ideally, we would like to choose $M$ large enough so that the optimal solution for (1) is not excluded. However, often such a value is not known a priori, and choosing a very large value for $M$ renders the MILP formulation more difficult to solve, since its integer relaxation becomes weak.

Often, the optimal solution of (11) is optimal for the original problem (1), and what remains is to prove that it is indeed optimal. To this end, we follow the approach proposed by Bai et al. (2013) and define an "outer problem" that consists of the original problem (1) with the added linear constraint

$$\sum_{j\in\mathcal{N}} w_j + \sum_{j\in\mathcal{N}} y_j \ge M. \tag{12}$$

The union of the feasible set of the outer problem and that of the big-$M$ MILP (11) (projected onto the $(x, y, w)$ space) includes the feasible set of the original problem (1). Therefore, the best among the optimal solutions of the two problems is guaranteed to be the optimal solution of (1).

Solving the outer problem is the context in which the logical Benders algorithm can be applied. For the remainder of this section we assume that the optimal objective function value $U$ for (1) is known. In case $U$ is not optimal, the algorithm is still able to determine the optimal solution.

## 2.3 Logical Benders within Branch-and-Bound Tree

We provide an interpretation of our logical Benders algorithm as a method to repeatedly restart a B&B algorithm. The algorithm chooses a leaf of the tree, solves that leaf, and searches for a sparse ancestor of that leaf that can also be fathomed. The tree is then reordered heuristically with the aim of placing fathomed nodes high in the tree. The next leaf is chosen to try to ensure it has a sparse ancestor with few fathomed descendants that shares as few fixed variables with the set of sparse fathomed nodes as possible.

It is recognized that the early branching decisions are crucial in branch and bound approaches to mixed integer programs, with strong branching often used to make the early branching decisions, which requires the solution of many linear programs for each branching decision. Branching choices that are deeper in the tree are made using cheaper approaches such as inference branching or using pseudocosts or a blend such as hybrid branching; see, for example, Achterberg (2007). Restarting the search can be used to try to improve the early branching decisions, exploiting preliminary computations in the construction of branching rules. Recent work on restarting a B&B search for an integer programming problem includes Kılınç-Karzan et al. (Karzan et al., 2009) and Fischetti and Monaci (Fischetti and Monaci, 2013, 2014). These references all run truncated B&B searches and then use the information gathered in these runs to set up branching rules to make a final complete run.

The reference Karzan et al. (2009) makes one initial incomplete branch and bound run until 200 nodes are fathomed. Sparse fathomed ancestors of these nodes are found by solving mixed integer programs. Branch-and-bound is then restarted, with branching guided by the set of sparse ancestors: given a particular node in the tree, the process is more likely to branch on a variable that appears in many of the sparse ancestors, with the corresponding constraints violated by the solution to the LP relaxation. The choice of branching node is determined by CPLEX in Karzan et al. (2009).

Fischetti and Monaci (Fischetti and Monaci, 2014) make five runs of B&B, with each run exploring at most five nodes. These runs all solve the initial LP relaxation, and then branch starting from different optimal solutions to the relaxation. One of these preliminary runs is then run to completion. Fischetti and Monaci (Fischetti and Monaci, 2013) try to identify a small set of important branching variables (a "backdoor"), where knowing the values of these variables would force many other variables to take particular values. The set of important variables is found by looking at good fractional solutions and ensuring that at least one of the fractional components appears in the backdoor. The branch-and-bound search is then restarted with branching priority given to the backdoor.

There has also been recent work on using machine learning techniques to

determine branching rules, for example to derive a branching rule that can replicate strong branching at lower computational cost by Alvarez et al. (Alvarez et al., 2017). Machine learning and related approaches to branching are surveyed by Lodi and Zarpellon (Lodi and Zarpellon, 2017).

Restarting branching algorithms for SAT has a longer history than for integer programming, dating back to Gomes et al. (Gomes et al., 1998). The DPLL algorithm for SAT fixes literals in a branching scheme and derives additional valid clauses using logical schemes such as resolution (Gomes et al., 2008). By restarting the process with these additional clauses, the branching decisions near the top of the tree are modified. It was observed in (Gomes et al., 1998) that this improves computational performance, in part because the algorithm does not invest too much time in a single branching ordering whose solution time might be an outlier. As noted by Huang (Huang, 2007), restarting with clause learning can be interpreted as a general resolution scheme and is now incorporated into most successful SAT algorithms. Backdoor branching was also originally proposed for SAT problems by Williams et al. (Williams et al., 2003).

Restarting is employed to improve the branching decisions high in the tree. Information from the results of prior branching decisions is used to guide the branching process after restarting. Our algorithm can be interpreted as restarting branch and bound at every iteration. The cited references restart far less frequently; Karzan et al. (2009); Fischetti and Monaci (2014, 2013) each restart at most a handful of times. An additional contrast between our approach and those of Karzan et al. (2009); Fischetti and Monaci (2014, 2013) is that we use restarting as a method to develop criteria for selecting the next node, whereas the cited references delegate node selection to CPLEX and instead focus on modifying the branching decisions employed at any given node of the tree to choose the branching variable.

Let $T$ be a fixed B&B tree as defined in Section 2, that is, all branching decisions are predetermined all the way to all leaves, and let $U$ be the optimal objective value of (1). We assume that the root node relaxation has a value worse than the upper bound, i.e., $\phi_P(\emptyset, \emptyset) < U$. Otherwise, we can terminate the procedure immediately, since $\phi_P(\emptyset, \emptyset)$ is always a lower bound on the optimal objective value. If $\phi_P(\emptyset, \emptyset) \geq U$, we can immediately conclude that $U$ is optimal.

Inspired by the terminology of the basic B&B algorithm described in Section 2.1, we call a node $[I^w, I^y]$ in $T$ a *fathomed node* if $\phi_P(I^w, I^y) \geq U$ and $\phi_P(\tilde{I}^w, \tilde{I}^y) < U$ for its parent $[\tilde{I}^w, \tilde{I}^y]$. A fathomed node is one for which the basic B&B algorithm would encounter one of the cases 1 or 2. The B&B algorithm would not explore the subtree below a fathomed node. On the other hand, the B&B algorithm would create children for any node $[\hat{I}^w, \hat{I}^y]$ above a fathomed node since no definite conclusion about the subtree below $[\hat{I}^w, \hat{I}^y]$ can be drawn based on the optimal solution of $[\hat{I}^w, \hat{I}^y]$ alone. Therefore, the fathomed nodes in a fixed tree $T$ is the minimal set of nodes that must be solved at some point in order to prove that $U$ is indeed the optimal objective value. Note that fathomed nodes depend exclusively on the structure of the tree $T$, and are independent of any algorithm.

Note that case 3 cannot occur since we assume that $U$ is the optimal objective value. Also, the tree $T$ with predetermined branching might not correspond to one that would be generated by the B&B algorithm. This is the case, for example, when child nodes are obtained by branching on a complementarity $j$ for which the optimal solution of the parent is already complementary. To avoid unnecessary work, the B&B algorithm only chooses branching complementarities for which the corresponding optimal values of the current node are not complementary, see case 4. In our context of interpreting the logical Benders method using a B&B tree, however, we permit this situation.

As described next, while the B&B algorithm is finding the fathomed nodes "from above" by branching from the root node to the fathomed nodes, we can interpret the logical Benders algorithm as a method that finds the fathomed nodes "from below".

Consider a fixed tree $T$. The logical Benders algorithm starts with an empty set of cuts $\mathcal{C} = \emptyset$, and therefore $\mathcal{F} = \{0,1\}^{n_c}$. It then chooses a piece $\bar{p} \in \mathcal{F}$, which corresponds to a leaf in the B&B tree. We now want to generate a cut $(I^w, I^y)$ so that

$$\sum_{i \in I^w} p_i + \sum_{i \in I^y} (1 - p_i) \geq 1 \tag{13}$$

excludes $\bar{p}$ as well as many other pieces, if possible. In our set notation, (13) is satisfied and the leaf $[J^w, J^y] = [I^w(\bar{p}), I^y(\bar{p})]$ corresponding to piece $\bar{p}$ is removed from $\mathcal{F}$ by the cut $(I^w, I^y)$ if and only if

$$I^w \subseteq J^w \text{ and } I^y \subseteq J^y. \tag{14}$$

One way to generate a cut consistent with the tree $T$ is presented in Algorithm 2.

Starting from the leaf corresponding to the piece $p$, this procedure works itself upwards in the tree. It continues along the path to the root until the parent $[\tilde{I}^w, \tilde{I}^y]$ of the current node $[I^w, I^y]$ has a value $\phi_P(\tilde{I}^w, \tilde{I}^y) < U$. In that case, $[I^w, I^y]$ must be a fathomed node. To express the fact that no solution better than $U$ can be found in any of the leafs (or pieces) below $[I^w, I^y]$, we add the corresponding cut $(I^w, I^y)$ to the master problem, so $\mathcal{C} \leftarrow \mathcal{C} \cup \{(I^w, I^y)\}$.

We now repeat this for the next iteration of the logical Benders decomposition, starting from any piece $p \in \mathcal{F}$ that has not yet been discarded by a cut. In the tree, this corresponds to any leaf that is not below a fathomed node. The algorithm above will again produce a new fathomed node that is added as a cut to the master problem. In this way, we will eventually discover all fathomed nodes in the tree. At that point, the feasible set $\mathcal{F}$ of the master problem will become empty, and the method concludes.

## 2.4 Minimal Cuts

In Section 1.2.2 we discussed the idea of sparsification. Given a valid cut $(I^w, I^y)$, i.e., $\phi_P(I^w, I^y) \geq U$, it might be possible to find smaller sets $\tilde{I}^w \subseteq I^w$ and $\tilde{I}^y \subseteq I^y$ with $\phi_P(\tilde{I}^w, \tilde{I}^y) \geq U$, so that $(\tilde{I}^w, \tilde{I}^y)$ still defines a valid cut. Such

**Algorithm 2** Find Fathomed Node

1: Input: Piece $p \in \{0,1\}^{n_c}$, path $P$ from the leaf $[I^w(p), I^y(p)]$ to the root node.

2: Let $j_1 \leftarrow j_2 \leftarrow \ldots \leftarrow j_{n_c}$ be the nodes along path $P$.

3: Set $[I^w, I^y] = [I^w(p), I^y(p)]$

4: **for** $l = 1, 2, 3, \ldots, n_c - 1$ **do**

5: Set $[\tilde{I}^w, \tilde{I}^y] = [I^w \setminus \{j_l\}, I^y \setminus \{j_l\}]$. ($[\tilde{I}^w, \tilde{I}^y]$ is the parent of $[I^w, I^y]$)

6: Solve (3) for $[\tilde{I}^w, \tilde{I}^y]$

7: **if** $\phi_P(\tilde{I}^w, \tilde{I}^y) < U$ **then**

8: Break (leave for loop)

9: Set $[I^w, I^y] \leftarrow [\tilde{I}^w, \tilde{I}^y]$

10: **end for**

11: Return $(I^w, I^y)$ as cut that identifies a fathomed node.

a sparser cut is preferable, since it excludes more pieces from $\mathcal{F}$, as can be seen from (13).

The hierarchy of cut sparsification defines a partial order, and we call the induced minimal elements *minimal cuts*. Within the algorithm described in the previous section, we can augment Algorithm 2 so that the procedure continues after the fathomed node has been found, see Algorithm 3. Whenever it turns out that removing a complementarity from the cut results in an invalid cut, it is simply added back, after which the search continues further along the path.

Clearly this procedure results in a minimal cut. It is important to note that the order in which the complementarities are released, i.e., the path (or tree), determines which particular minimal cut is found.

The set of minimal cuts does not depend on the choice of a particular tree, it is defined by the problem statement together with the upper bound $U$. Consider a minimal cut $(I^w, I^y)$. We can now construct a tree in which $[I^w, I^y]$ is a fathomed node, simply by choosing the complementarities in $I^w \cup I^y$ as the branching decisions from the root node to $[I^w, I^y]$. On the other hand, given the index sets $I^w$ and $I^y$ from the minimal cut, in a different given tree $\tilde{T}$, there might not be a node $[I^w, I^y]$ with the same index sets. In that case, the cut given by $(I^w, I^y)$ might correspond to more than one fathomed node in $\tilde{T}$, each one has a path to the root that includes all of the complementarities $I^w \cup I^y$. In fact, given a set of minimal cuts $\{I^w_k, I^y_k\}_{k=1}^K$, there might not exist a tree $T$ in which those cuts correspond to nodes of the form $[I^w_k, I^y_k]$. We illustrate these observations with an example:

**Algorithm 3** Cut Sparsification

1: Input: Piece $p \in \{0,1\}^{n_c}$, path $P$ from the leaf $[I^w(p), I^y(p)]$ to the root node.

2: Let $j_1 \leftarrow j_2 \leftarrow \ldots \leftarrow j_{n_c}$ be the nodes along path $P$.

3: Set $[I^w, I^y] = [I^w(p), I^w(p)]$

4: **for** $l = 1, 2, 3, \ldots, n_c$ **do**

5: Set $[\tilde{I}^w, \tilde{I}^y] = [I^w \setminus \{j_l\}, I^y \setminus \{j_l\}]$. ($[\tilde{I}^w, \tilde{I}^y]$ is the parent of $[I^w, I^y]$)

6: Solve (3) for $[\tilde{I}^w, \tilde{I}^y]$

7: **if** $\phi_P(\tilde{I}^w, \tilde{I}^y) \geq U$ **then**

8: Set $[I^w, I^y] \leftarrow [\tilde{I}^w, \tilde{I}^y]$

9: **end for**

10: Return $(I^w, I^y)$ as minimal cut.

**Example 2.1** *Consider the following LPCC:*

$$\begin{array}{llll} \textit{minimize} & -2x_1 - x_2 - x_3 & & \\ \textit{subject to} & x_i & \leq 4 & i = 1,2,3 \\ & x_i & = y_i & i = 1,2,3 \\ & \sum_{k \neq i} x_k - 3x_i + 6 & = w_i & i = 1,2,3 \\ 0 \leq & y \perp w & \geq 0 & \end{array} \tag{15}$$

*and the set of cuts* $(I_1^w, I_1^y) = (\emptyset, \{1\})$, $(I_2^w, I_2^y) = (\emptyset, \{2\})$ *and* $(I_3^w, I_3^y) = (\{1,2,3\}, \emptyset)$. *One solution to this problem is* $x_1 = x_2 = y_1 = y_2 = 3$ *and* $w_3 = 12$ *with all other variables set to zero. The optimal value of the LPCC is* $U = -9$ *and that of the relaxation is* $\phi_P(\emptyset, \emptyset) = -16$. *We have* $\phi_P(I_1^w, I_1^y) = -6$ *and* $\phi_P(I_2^w, I_2^y) = -9$, *hence* $(I_1^w, I_1^y)$ *and* $(I_2^w, I_2^y)$ *are minimal cuts. We also have* $\phi_P(I_3^w, I_3^y) = \infty$ *(infeasible) and* $\phi_P(\{1,2\}, \emptyset) = -14$, $\phi_P(\{1,3\}, \emptyset) = -14$ *and* $\phi_P(\{2,3\}, \emptyset) = -12$, *so* $(I_3^w, I_3^y)$ *is also minimal.*

*Let* $T$ *be any B&B tree for this problem. If the first branching complementarity was component* $1$, *then the node* $[\emptyset, \{2\}]$ *corresponding to cut* $(I_2^w, I_2^y)$ *does not exist. We can follow the same reasoning for any other first branching complementarity. Therefore, for every tree there exists at least one of these minimal cuts which does not correspond to a node.*

## 2.5 Nodes in a Branch-and-Bound Tree

Let $T$ be a fixed tree, and $\mathcal{C} = \{(I_k^w, I_k^y) : k = 1, \ldots, K\}$ be a set of cuts. We can subdivide the nodes of $T$ into three different classes. Let $[J^w, J^y]$ be a node

in $T$.

- We say that node $[J^w, J^y]$ is *discarded* if there exists a cut $(I_k^w, I_k^y) \in \mathcal{C}$ so that $I_k^w \subseteq J^w$ and $I_k^y \subseteq J^y$ with at least one of the two inclusions being strict. In this case, the node $[J^w, J^y]$ lies below a fathomed node identified by $(I_k^w, I_k^y)$. All the leaves in the subtree below $[J^w, J^y]$ correspond to pieces that are already excluded in $\mathcal{F}$.
- We say that node $[J^w, J^y]$ is *explored* if there exists a fathomed node corresponding to some cut $(I_k^w, I_k^y) \in \mathcal{C}$ so that $[J^w, J^y]$ is on the path from the fathomed node to the root node. In particular, fathomed nodes are explored.
- We say that node $[J^w, J^y]$ is *unexplored* if it is not discarded or explored. In this case, $[J^w, J^y]$ does not lie below a fathomed node or on the path to a fathomed node identified by any of the cuts. All the leaves in the subtree below an unexplored node correspond to pieces that are still in the set $\mathcal{F}$. An unexplored node has the potential to be a fathomed node, and finding a minimal cut identifying it would remove all such pieces from $\mathcal{F}$.

Furthermore, we will call an unexplored node an *open* node, if its parent is explored. Among the unexplored nodes, open nodes have the potential to generate cuts that remove the most pieces (leaves) in that part of the tree if they turn out to be fathomed nodes.

Let us demonstrate this in an example.

**Example 2.2** *Using the same problem as in Example 2.1, consider the tree in Figure 1 with the cuts* $(I_1^w, I_1^y)$ *and* $(I_2^w, I_2^y)$*. The fathomed nodes corresponding to these cuts are 3 and 5, which correspond to* $[J_1^w, J_1^y] = [\emptyset, \{1\}]$ *and* $[J_2^w, J_2^y] = [\{1\}, \{2\}]$*, respectively, according to our notation. Notice that* $[J_1^w, J_1^y] = [I_1^w, I_1^y]$*, but node* $[I_2^w, I_2^y]$ *does not exist in this tree.*

*Nodes 8, 9,...,15 are discarded nodes, while nodes 1, 2, 3 and 5 are explored nodes. Node 4, 6 and 7 are unexplored nodes and therefore 4 is the only open node.*

*Now, if we consider the tree in Figure 2 the fathomed nodes correspond to 5, 7, 9 and 13 and the open nodes are 6 and 12.*

## 2.6 Constructing a Tree

The initial version of the B&B-based logical Benders algorithm described in Section 2.3 assumed that the tree $T$ was given and fixed throughout the procedure. Clearly, the amount of work, when measured in the number of (master problem) iterations, depends on the choice of the tree, and in particular on the number of fathomed nodes in that tree, since the algorithm can only terminate when they have all been identified.

We are usually not given a tree a priori that results in good performance for a given problem. The algorithm we propose here is related to the idea of

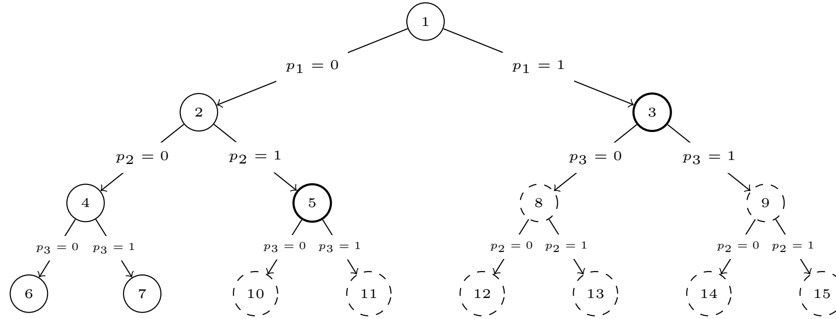

Figure 1: Classes of nodes in a B&B tree

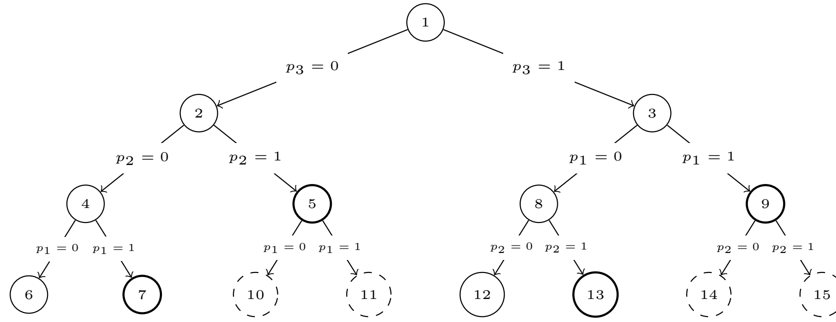

Figure 2: Classes of nodes in a B&B tree

information-based branching presented by Karzan et al. (2009). Our approach follows the same spirit, but in a dynamic way. It constructs a "working tree" as a preliminary choice of the final tree in each iteration, taking into account the cuts that have been generated so far. It picks a new piece $p$, and then executes the cut sparsification Algorithm 3 to generate a new cut. It is important to note that this tree is never assembled explicitly, it is only a conceptual device that helps us to identify the input for Algorithm 3, i.e., the piece $p$ and the path $P$.

Our algorithm for choosing $p$ and $P$ has two steps. It first (virtually) constructs a working tree and finds an open node in that tree, and then chooses a piece (or leaf) under the open node. For this, the working tree only needs to be defined up to the open and fathomed nodes. Algorithm 4 gives the framework for recursively defining a working tree. It proceeds recursively from the root node, calling itself for the generation of the subtrees after branching on a complementarity. At the beginning it is called with the root node and all available cuts, i.e., TREE FORMATION($\{(I_k^w, I_k^y)\}_{k=1}^K$, $[\emptyset, \emptyset]$). The recursion is set up in a way so that only cuts are passed to the next level that are still relevant for the generation of the subtree rooted at the incoming node, in the sense that they could lead to a fathomed node. If no such cuts are available, the current node is marked as open in Step 2. On the other hand, if there is a cut that is

**Algorithm 4** Procedure for generating working tree

$T = $ TREE FORMATION($\{(I_k^w, I_k^y)\}_{k=1}^K$, $[J^w, J^y]$)

INPUT: Cuts $\{(I_k^w, I_k^y)\}_{k=1}^K$. Root node $[J^w, J^y]$ of subtree to be constructed.

1: **if** $K = 0$ **then**
2: Label $[J^w, J^y]$ as *open* node. **return.**
3: **if** $I_k^w \subseteq J^w$ and $I_k^y \subseteq J^y$ for some $k$ **then** ▷ A cut is violated
4: Label $[J^w, J^y]$ as *fathomed* node. **return.**
5: Label $[J^w, J^y]$ as *explored* node.
6: $\tilde{\mathcal{N}} = \{i \in \mathcal{N} : i \in (I_k^w \cup I_k^y) \setminus (J^w \cup J^y)$ for some $k = 1, \ldots, K\}$.
7: **if** (Strategy 1) **then**
8: $v_i := \#\{(I_k^w, I_k^y) : i \in I_k^w \cup I_k^y$ for some $k = 1, \ldots, K\}$ for all $i \in \tilde{\mathcal{N}}$
9: Select $j \in \operatorname{argmax}_{i \in \tilde{\mathcal{N}}}\{v_i\}$.
10: **else** (Strategy 2)
11: Let $s_k = \#\{(I_k^w \cup I_k^y) \setminus (J^w \cup J^y)\}$ for all $k$.
12: Let $\bar{s} = \min_k\{s_k\}$.
13: $v_i := \#\{(I_k^w, I_k^y) : i \in I_k^w \cup I_k^y$ and $s_k = \bar{s}$ for some $k = 1, \ldots, K\}$ for all $i \in \tilde{\mathcal{N}}$
14: Select $j \in \operatorname{argmax}_{i \in \tilde{\mathcal{N}}}\{v_i\}$.
15: $\mathcal{I}^w := \{(I_k^w, I_k^y) : j \in I_k^y, k = 1, \ldots, K\}$.
16: $\mathcal{I}^y := \{(I_k^w, I_k^y) : j \in I_k^w, k = 1, \ldots, K\}$.
17: **if** $\#\mathcal{I}^w < \#\mathcal{I}^y$ **then**
18: Construct subtree for $w$-child: Call TREE FORMATION($\mathcal{I}^w$, $[J^w \cup \{j\}, J^y]$).
19: Construct subtree for $y$-child: Call TREE FORMATION($\mathcal{I}^y$, $[J^w, J^y \cup \{j\}]$).
20: **else**
21: Construct subtree for $y$-child: Call TREE FORMATION($\mathcal{I}^y$, $[J^w, J^y \cup \{j\}]$).
22: Construct subtree for $w$-child: Call TREE FORMATION($\mathcal{I}^w$, $[J^w \cup \{j\}, J^y]$).
23: **return.**

violated by the incoming node (see also (14)), then the incoming node is marked as fathomed. If neither of those conditions are met, a branching complementarity needs to be chosen among those that have not yet been branched on (in the set $\tilde{\mathcal{N}}$). The choice of the branching variable $j$ is guided by the following observations.

1. An open node $[J^w, J^y]$ has the potential to become a fathomed node, namely when $\phi_P(J^w, J^y) \geq U$. In that case, a single iteration of the algorithm suffices to generate a cut to remove all leafs under $[J^y, J^w]$. So, in the most optimistic outcome, the remaining number of iterations is equal to the number of open nodes in the current B&B tree. Therefore, we aim at constructing a working tree with a small number of open nodes.

2. For every open node there is a fathomed node at the same or lower level since its parent is explored. We therefore would like to generate a tree in which the fathomed nodes are high up in the tree. This way, if the open node can be immediately fathomed, the higher in the tree, the sparser it will be.

3. The fathomed nodes that are identified by a given cut of cardinality $l$ are at least $l$ levels down. Therefore, in the best case possible, all fathomed nodes should be on the same level as the cardinality of the cut they are represented with.

4. A complementarity that appears in no cut should not be chosen for branching. Otherwise, all fathomed and open nodes would be pushed down by one level, counteracting the goal described in observation (2) above.

We explore two different branching strategies. Strategy 1 chooses a complementarity that appears in most of the cuts that are still relevant. In this way, we hope to keep the level of the deepest fathomed node small, see observation 3 above.

The second strategy is based on the following observation: Suppose that there is a cut that includes only one complementarity that has not yet been branched on. Choosing this complementarity for branching would create one child node that can immediately be fathomed, and we effectively reduced the complexity of the remaining subtree by one complementarity in one shot. This observation motivates us to give priority to the sparsest cuts, i.e., those that include the smallest number $\bar{s}$ of remaining complementarities. Steps 11–12 choose one of the complementarities that appears most often in the sparsest cuts. In both strategies, if more than one complementarity satisfies the criteria above, then we choose the one that appears first in a fixed priority list. In our implementation, this list starts as the components, sorted in a descending order according to their complementarity violation, obtained from solving the root node. The list is updated as the tree becomes constructed along the path from the root to last chosen leaf.

Before calling the algorithm recursively to generate the subtree corresponding to the new child nodes, Steps 15 and 16 remove the cuts that are irrelevant in the respective subtree. Finally, the subtrees are generated.

## 2.7 Choosing an open node

Now that we specified how we define the working tree based on the cuts in a given iteration, we need to choose one of its open nodes in order to generate a new cut. We do this by executing a variation of the tree generation algorithm in the previous section.

In Algorithm 4, we set up the recursion in steps 17–22 so that the subtree with the fewest remaining cuts is generated first. While this makes no difference for the final working tree that is formed, it determines the order in which we encounter the open nodes. Since fewer cuts typically lead to subtrees in which the fathomed and open nodes are closer to its root, we are likely to find an open node quickly that is high up in the tree and has the potential to result in a sparse cut. The search procedure executes Algorithm (4) and returns the first open node that it encounters. Since the order of exploration prioritizes smaller subtrees, this depth-first search is likely to find an open node quickly, close to the root.

## 2.8 Choosing a piece

Once an open node $[J^w, J^y]$ has been determined, the algorithm requires a piece $p$ and a path $P$ as input for the cut generation procedure in Algorithm 3. To be consistent with the current working tree, the path between $[J^w, J^y]$ and the root node is chosen according to the working tree determined by Algorithm 4. It remains to choose a leaf corresponding to $p$ that lies below $[J^w, J^y]$, and the path between that leaf and $[J^w, J^y]$.

Our choice is guided by the observation that Algorithm 3 removes a complementarity whenever $\phi_P(\tilde{I}^w, \tilde{I}^y) \geq U$, where $(\tilde{I}^w, \tilde{I}^y)$ is the current trial cut in Algorithm 3. This suggests that it is beneficial to choose a leaf $p$ that has a large value of $\phi_P$ or is infeasible.

To determine such a piece, we solve the relaxation (3) corresponding to $\phi_P(J^w, J^y)$. If $\phi_P(J^w, J^y) \geq U$, then the open node $[J^w, J^y]$ is actually a fathomed node, and we can generate a cut corresponding to it. Formally, we choose any piece $p$ below $[J^w, J^y]$ and any path from that piece to $[J^w, J^y]$. The first iterations of Algorithm 3 will then remove all complementarities $j_l$ that are not in $J^w \cup J^y$. In practice, we start Algorithm 3 from $(I^w, I^y) = (J^w, J^y)$ and avoid the unnecessary steps.

If $\phi_P(J^w, J^y) < U$, then the relaxation for $[J^w, J^y]$ must be feasible and has an optimal solution $(\tilde{x}, \tilde{w}, \tilde{y})$. We set, for all $i \in \mathcal{N} \setminus (J^w \cup J^y)$ (i.e., all complementarities that have not yet been branched on) $r_i = \min\{\tilde{w}_i, \tilde{y}_i\}$ and order them in decreasing order. We now choose the complementarities in that order, pick the $w$-branch toward the leaf if $w_i > y_i$ and the $y$-branch otherwise. If there is a tie and $w_i = y_i > 0$, we pick from the pre-chosen order, and if there

is a tie with $w_i = y_i = 0$, we choose the side with the larger multiplier. This gives us the path and the leaf $p$.

# 3 Enhancements of the Logical Benders Algorithm

Recall that the logical Benders algorithm consists of two basic steps: Choose a piece $p \in \mathcal{F}$, and generate a cut $(I^w, I^y)$ that excludes $p$ from $\mathcal{F}$. For each of them, we consider different options, and we will compare their numerical performance in Section 4.

## 3.1 Piece Selection

In (Bai et al., 2013), the authors find the new piece to explore using a black-box satisfiability problem (SAT) solver, part of the SIMULINK packages available in MATLAB. This approach has the disadvantage that it blindly selects a piece satisfying all cuts without considering the implicit information embedded in these cuts. The methods described in Sections 2.7 and 2.8 offer an alternative. Because it is based on a working tree it can choose a piece (or an open node) that has the potential to lead to a cut that is quite sparse. More importantly, this procedure generates cuts that are consistent with the existing cuts in the sense that it removes potentially many additional pieces from $\mathcal{F}$ instead of being redundant. Furthermore, if the open node is fathomable it saves any additional exploration on the subtree emerging from it as the original method would do. This advantage of discarding open nodes instead of leaves is demonstrated in the numerical results section. We will refer to the piece selection procedure described in Section 2.8 as the "tree-guided piece selection".

## 3.2 Generation of Sparse Cuts

In this section we describe and discuss three different approaches to obtain sparse cuts which allow us to remove pieces that need not be explored. Cuts are constructed in a way such that any piece $p$ removed by it is guaranteed to satisfy $\phi_P(p) \geq U$. Recall that a cut is an inequality determined by two disjoint subsets $I^w, I^y \subseteq \{1, \ldots, n_c\}$ of the form

$$\sum_{i \in I^w} p_i + \sum_{i \in I^y} (1 - p_i) \geq 1, \tag{16}$$

hence sparsifying a cut translates into finding smaller sets $\bar{I}^w$ and $\bar{I}^y$ with a certificate that the primal relaxation $\phi_P(\bar{I}^w, \bar{I}^y) \geq U$.

The first approach, already introduced in Section 2.7, evaluates primal relaxations in a sequential order given by a "virtual" B&B tree. The second one, formulates an $\ell_1$-norm minimization problem over the dual feasible region of the explored piece. Finally, we propose a third alternative that combines

the advantages of the first two options. A detailed description of the methods follows.

### 3.2.1 Path-based Procedure

The general steps for the second sparsification method have been described in Algorithm 3. Given a piece $p$ and a predefined order of indices $j_1 \leftarrow j_2 \leftarrow j_3 \leftarrow \ldots \leftarrow j_{n_c}$, we sequentially solve relaxations of this piece following the order. Let us illustrate this procedure through an example.

**Example 3.1** *Using the same problem as in example (2.1)*

$$\begin{array}{llll} \textit{minimize} & -2x_1 - x_2 - x_3 & & \\ \textit{subject to} & x_i & \leq 4 & i = 1,2,3 \\ & x_i & = y_i & i = 1,2,3 \\ & \sum_{k \neq i} x_k - 3x_i + 6 & = w_i & i = 1,2,3 \\ 0 \leq & y \perp w & \geq 0, & \end{array} \tag{17}$$

*suppose we are given piece* $p = (1,0,0)$*, that is* $I^w = \{2,3\}$ *and* $I^y = \{1\}$*, and the path from leaf to root is given by* $3 \to 2 \to 1$*. We are also given the optimal upper bound* $U = -9$*. Notice that* $\phi_P(p) = -6$*, so the piece could be immediately removed with the cut* $(I^w, I^y)$*, but this cut will remove only p and no other piece. The sequential sparsification procedure checks if we can do better. It first tries the relaxation* $\phi_P(\{2\},\{1\})$*, which has a value of* $-6$*, so now we found a better cut. It then continues with* $\phi_P(\emptyset,\{1\})$*, again with a value of* $-6$*, so the cut has been sparsified even further. It ends by trying out* $\phi_P(\emptyset,\emptyset)$ *(i.e., the full relaxation) with a value of* $-16$*. This means that complementarity* $y_1 = 0$ *cannot be relaxed so index* $i = 1$ *remains on* $I^y$*. The final cut is therefore* $(\emptyset,\{1\})$*. If we were given the piece* $p = (0,0,1)$*, with same path* $3 \to 2 \to 1$*, the obtained cut will be* $(\emptyset,\{3\})$ *due to symmetry of this example problem. This example is depicted in Figure 3. Dashed and bold lines represent accepted and rejected relaxations of complementarities, respectively. Bold circles correspond to fathomed nodes. Everything grayed out are nodes and branches not observed yet.*

Clearly, by construction, every cut $(I^w, I^y)$ obtained from this procedure is minimal.

This method can take advantage of a situation in which the tree-guided piece selection finds an open node $[I^w, I^y]$ with $\phi_P(I^w, I^y) \geq U$. It is clear that Algorithm 3 removes all indices along the path from the leaf to $[I^w, I^y]$, and so we save iterations by running Algorithm 3 with an abbreviated path that starts at $[I^w, I^y]$ instead of the leaf.

As mentioned earlier, this method guarantees that the resulting cut $(I^w, I^y)$ is *minimal.* It could happen that node $[I^w, I^y]$ does not exist in the current working tree, which can be interpreted as if some branching decisions in the

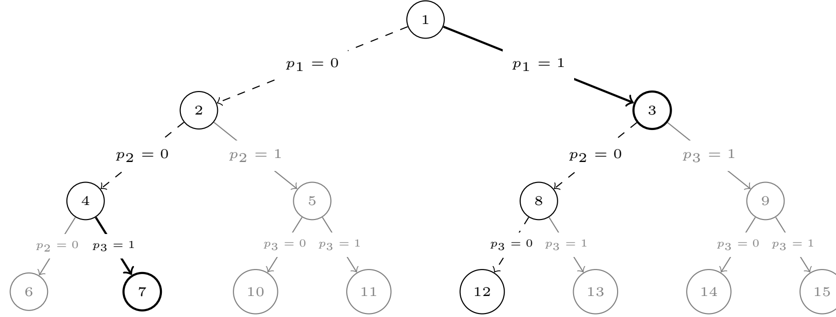


Figure 3: Sequential sparsification in a B&B tree

tree were not necessary, and hence some fathomed nodes appear in levels lower than desired. This motivates the procedure for the tree to be updated in every iteration, so that fathomed nodes (and consequently open nodes) might be moved higher in the tree.

### 3.2.2 Optimization-based procedure

Let us first assume that a *feasible* piece $p$ has been selected from $\mathcal{F}$ by the piece selection procedure. As mentioned in Section 1.2.2, cuts are closely related to the dual variables $\lambda^w$ and $\lambda^y$ in (8). A valid cut is obtained with $I^w = \{i : \lambda_i^w > 0\}$ and $I^y = \{i : \lambda_i^y > 0\}$. Hence, an alternative to find sparse cuts is to solve the following linear program:

$$\underset{(\mu_I,\, \mu_E,\, \lambda^w,\, \lambda^y) \in \Omega_D(\bar{I}^w, \bar{I}^y)}{\text{minimize}} \sum_{i \in \bar{I}^w} \lambda_i^w + \sum_{i \in \bar{I}^y} \lambda_i^y, \tag{18}$$

where $\bar{I}^w = \bar{I}^w(p)$ and $\bar{I}^y = \bar{I}^y(p)$. A solution $(\hat{\lambda}^y, \hat{\lambda}^w)$ of this problem provides a valid cut $(\hat{I}^w, \hat{I}^y)$ with $I^y := \{i : \hat{\lambda}_i^y > 0\}$ and $I^w := \{i : \hat{\lambda}_i^w > 0\}$, and its corresponding relaxation has a value greater or equal than $U$. This condition is forced in the $-b_I^T \mu_I + b_E^T \mu_E \geq U$ constraint, in the description of $\Omega_D(I^w, I^y)$. The objective function is interpreted as the $\ell_1$-norm of variables $\lambda^w$ and $\lambda^y$ which intends to steer their components towards zero. In a strict sense, to find a sparsest solution we should be using the $\ell_0$-norm instead in the objective, but then problem (18) becomes highly non-convex. Following (Candes et al., 2008), we enhance the solution of (18) with an iterative re-weighting scheme. Given a set of initial weights $\omega^0 \in \mathbb{R}_+^{|I^w|}$ and $\gamma^0 \in \mathbb{R}_+^{|I^y|}$, we set the iteration counter $k = 0$ and solve the problem

$$\underset{(\mu_I,\, \mu_E,\, \lambda^w,\, \lambda^y) \in \Omega_D(I^w, I^y)}{\text{minimize}} \sum_{i \in I^w} \omega_i^k \lambda_i^w + \sum_{i \in I^y} \gamma_i^k \lambda_i^y, \tag{19}$$

and with the optimal solution $(\hat{\mu}_I^k, \hat{\mu}_E^k, (\hat{\lambda}^w)^k, (\hat{\lambda}^y)^k)$ of (19) we update the weights $\omega_i^{k+1} = \frac{1}{\max\{\varepsilon, (\hat{\lambda}^w)_i^k\}}$ and $\gamma_i^{k+1} = \frac{1}{\max\{\varepsilon, (\hat{\lambda}^y)_i^k\}}$, set $k = k+1$ and re-solve. The $\varepsilon$ parameter serves to ensure the weights are well defined in the

circumstance that $\lambda$ becomes zero. In our implementations, $\varepsilon$ was set to $10^{-6}$. This iterative procedure gives a variable which was close to zero in one iteration a larger weight in the next one and therefore it is more likely to be pushed to zero. The method ends when two consecutive iterates are equal. This procedure can be interpreted, as explained in (Candes et al., 2008), as sequentially minimizing linearizations of $\sum_i \log(\lambda_i)$ over $\Omega_D$ and, hence, the closer to zero a variable is, the steepest its slope becomes. In our experiments, this procedure converges very quickly, requiring no more than 6 iterations even in the largest instances. We set the initial weights to 1 for every component of $\lambda^w$ and $\lambda^y$.

Under the circumstance that the tree-guided piece selection finds an open node $[I^w, I^y]$ with $\phi_P(I^w, I^y) \geq U$, we start the procedure directly with the index sets $I^w$ and $I^y$, provided the corresponding relaxation $\Omega_D(I^w, I^y)$ is feasible.

Now consider the case in which the primal of the selected piece $p \in \mathcal{F}$ is infeasible. In case the dual problem (8) is feasible, we can still follow the procedure above. However, if (8) is infeasible, we formulate the $\ell_1$-norm minimization problem over the homogenized set $\Omega_{D_0}(I^w, I^y)$. Once the weighted iterative heuristic finishes in an unbounded ray $(\hat{\mu}_I, \hat{\mu}_E, \hat{\lambda}^w, \hat{\lambda}^y)$, we check whether the corresponding primal relaxation became feasible. If not we set $I^y = \{i : \hat{\lambda}_i^y > 0\}$ and $I^w = \{i : \hat{\lambda}_i^w > 0\}$ as the index sets of the newly found sparse cut. Otherwise, we continue the $\ell_1$-norm minimization on the corresponding dual of the relaxed primal.

Although this method has the nice feature that it requires the solutions of just a few LPs, it has no guarantee that it will produce a minimal cut. Consider the following very simple example:

$$\begin{array}{rl} \min_\lambda & \lambda_1 + \lambda_2 \\ \text{subject to} & \lambda_1 + 2\lambda_2 \geq 3 \\ & 2\lambda_1 + \lambda_2 \geq 3 \\ & \lambda \geq 0. \end{array} \tag{20}$$

If the initial weights are set to one, the procedure will converge in the second iteration to $(1, 1)$, although the sparsest solutions are $(3, 0)$ and $(0, 3)$.

This gives way for a third procedure to generate sparse cuts: the Hybrid method.

### 3.2.3 Hybrid procedure

Both presented sparsification procedures have their own advantages and downsides. The $\ell_1$-norm minimization approach finds cuts quickly by iteratively solving linear programs, which can even be hot-started in a primal simplex algorithm since the feasible region does not change. But there is no guarantee that the resulting cuts are minimal. On the other hand, the sequential procedure finds minimal cuts, but it requires $n_c$ LP solves, making it too costly as the dimension of the complementarities increases, even if solved with the dual simplex method to use hot starts as we did in our implementation.

**Algorithm 5** Hybrid Sparsification

1: Input: Piece $p \in \{0,1\}^{n_c}$, a path $P$.
2: Set $(I^w, I^y) = (I^w(p), I^w(p))$
3: Solve problem (19) to obtain solution $(\hat{\lambda}^w, \hat{\lambda}^y)$.
4: Set $I^w := \{i : \hat{\lambda}_i^w > 0\}$ and $I^y := \{i : \hat{\lambda}_i^y > 0\}$.
5: Let $j_1 \leftarrow j_2 \leftarrow \ldots \leftarrow j_L$ be the subset of the path $P$, such that $\{j_i\}_{i=1}^L = I^w \cup I^y$.
6: **for** $l = 1, 2, 3, \ldots, L$ **do**
7: Set $(\tilde{I}^w, \tilde{I}^y) = (I^w \setminus \{j_l\}, I^y \setminus \{j_l\})$. ($(\tilde{I}^w, \tilde{I}^y)$ is the parent of $(I^w, I^y)$)
8: Solve (3) for $(\tilde{I}^w, \tilde{I}^y)$
9: **if** $\phi_P(\tilde{I}^w, \tilde{I}^y) \geq U$ **then**
10: Set $(I^w, I^y) \leftarrow (\tilde{I}^w, \tilde{I}^y)$
11: **end for**
12: Return $(I^w, I^y)$ as minimal cut.

We can exploit the advantages of both methods in a hybrid sparsification procedure. The approach consists of two steps: (1) the $\ell_1$-norm minimization sparsification routine is called to compute a cut $(I^w, I^y)$ given by $I^w := \{i : \hat{\lambda}_i^w > 0\}$ and $I^y := \{i : \hat{\lambda}_i^y > 0\}$, where $\hat{\lambda}^w$ and $\hat{\lambda}^y$ are the outputs of the iterative re-weighting scheme, and (2) the sequential sparsification steps along the path $P$ are executed, assuming that the complementarities in $I^w \cup I^y$ can be directly removed in Algorithm 3 without computations.

The underlying idea is that the $\ell_1$-norm minimization already removes many complementarities from the cut, so that the sequential procedure does not need to go through all components from the leaf to the root. A formal outline of the procedure is described in Algorithm 5

## 3.3 Remarks

We finish this section by highlighting some observations regarding the power of the presented cut sparsification procedures.

- Throughout much of this paper, as stated in section 2.2, we assumed we are given the optimal value $U$ of the LPCC and that the logical Benders algorithm is used to certify optimality. It is only in this setting where we can claim that the cuts generated by the sequential or hybrid methods are actually minimal cuts, where minimality is considered with respect to the actual optimal value of (1). This is clear by how these methods were designed. Under the circumstance that the provided incumbent is

not the optimal solution, the methods described in the previous sections can be adopted. The only difference is that the upper bound needs to be updated whenever a piece $p \in \mathcal{F}$ is selected such that $\phi_P(p) < U$. Whenever these updates occur the cuts generated so far could potentially be sparsified further, by replacing the newly found incumbent as value to be comparing against, in $\Omega_D$ and Algorithm 3. Keep in mind, though, that if resparsification of a cut is called, for example, in the sequential or hybrid method, the path from leaf to root (or subset of the path, to be more precise) to be considered will be from the most recent "virtual" tree and not the one used when the corresponding cut was generated. In this paper, we do not consider resparsification and its effectiveness remains to be investigated later.

- If the tree $T$ is fixed throughout the whole algorithm, that is, the sequences of branching decisions from every leaf to the root do not change, then the order in which the pieces (leaves) are selected in the procedure is irrelevant. The set of cuts at the end of the method is independent of this selection.

# 4 Numerical Results

This section contains various numerical experiments in order to demonstrate the effectiveness and robustness of the proposed method. We solved instances with complementarity dimension $n_c$ ranging from 100 to 1,000 and compared four different variants of the logical Benders methods: (1) Base, (2) Splitting ("Spl"), (3) Optimization-based ("$\ell_1$"), and (4) hybrid between Optimization and Path-based ("Hyb"). Variant (1) refers to the baseline logical Benders method. That is, pieces are selected via a black-box SAT solver and cuts are strengthened with the Splitting sparsification. All remaining variants use the Tree-guided node selection.

By Splitting sparsification we refer to the method described by the end of section (1.2.1). An initial cut $(I^w, I^y)$ is obtained from the optimal solution of the dual piece (8), as described in section 1.2.2. The dual vectors, $\lambda^y$ and $\lambda^w$, are sorted in descending order and the top half of the non-zero components of each vector are selected to form two sets $\tilde{I}^w$ and $\tilde{I}^y$. If $\phi_P(\tilde{I}^w, \tilde{I}^y) \geq U$, then $I^w := \tilde{I}^w$ and $I^y := \tilde{I}^y$ and the sparsification method continues. Otherwise, the returned cut is $(I^w, I^y)$.

As for the tree guided node selection, it is set up so that if the open node $[I^w, I^y]$ is "fathomable", meaning that its primal value lies above $U$, we follow Algorithm 2. That is, we keep the cut $(I^w, I^y)$ as is. This can be interpreted as if we accept the tree that has been constructed so far, and prefer to explore other nodes instead of trying to restructure it.

The tested instances were generated following the steps described in Hu et al.

(2008). This reference used a slightly different structured LPCC, namely

$$\begin{array}{llll} \underset{x\geq 0, y}{\text{minimize}} & & c^T x + d^T y & \\ \text{subject to} & & Ax + By & \geq f \\ & 0 \leq & y \perp q + Nx + My & \geq 0. \end{array} \tag{21}$$

The procedure to generate instances is described below. After generating instances of this type, converting to the structure of (1) is straightforward.

**Instance Generator**[1]

INPUT: Dimensions $n$, $m$ and $k$, for $x$, $y$ and $f$, respectively. Density value $s$.

OUTPUT: Matrices $c$, $d$, $A$, $B$, $f$, $M$, $N$ and $q$.

1: Generate $x \sim N_n(0, 1)$. Set $x = |x|$.
2: Generate $y \sim N_m(0, 1)$. Set $y_i = 0$ if $y_i < 0$.
3: Generate $c \sim U_n(0, 1)$ and $d \sim U_m(1, 3)$
4: Generate $A \sim U_{k\times n}(0, 1)$ and $B \sim U_{k\times m}(0, 1)$ with density $s$.
5: Generate $r \sim DU(\{0, 1, \ldots, m\})$
6: Let $s_M = \frac{2000-m}{m^2}$
7: Generate $E \sim U_{r\times(m-r)}(-1, 1)$ with density $s_M$.
8: Generate $d_1 \sim U_r(0, 2)$ and $d_2 \sim U_{m-r}(0, 2)$.
9: Let $D_1 = \text{diag}(d_1)$ and $D_2 = \text{diag}(d_2)$.
10: Let $M = \begin{pmatrix} D_1 & E \\ -E & D_2 \end{pmatrix}$
11: Generate $N \sim U_{m\times n}(-1, 1)$.
12: Generate $q \sim U_m(-20, -10)$
13: Let $f = Ax + By - |\varepsilon|$, with $\varepsilon \sim N_k(0, 1)$.

The dimensions of our test instances follow similar relations as in Hu et al. (2008): $[n, m, k] = [100, 100, 90]$, $[300, 300, 200]$ and $[500, 500, 450]$. For the largest test set we chose $[n, m, k] = [1000, 1000, 400]$.

During our experiments we noticed some patterns which seemed to make a problem harder or easier. For example, if coupling constraints were present (i.e. $B \neq 0$), the method required more iterations on average to be solved, for a fixed dimension. Therefore, as part of our examination, we consider both the cases with and without coupling constraints (by either setting $B \sim U_{k\times m}(0, 1)$ or $B = 0$ in the Instance Generator).

As explained in Section 2.2, we obtain a candidate optimal solution by solving problem (11) first and then attempt to provide a certificate of optimality by solving the "outer problem". In this setting, we can immediately terminate if

[1] Notation: $N_n(0, 1)$, refers to $n$ i.i.d. draws from a standard normal distribution; $U_n(a, b)$, $n$ draws from a uniform distribution in the $(a, b)$ interval, and $DU(\{a_1, \ldots, a_m\})$, a uniform draw from the set $\{a_1, \ldots, a_m\}$.

the objective value is lower than the relaxation of the outer problem (that is, (1) with the constraint (12)). Therefore, we chose the big-$M$ parameter in a way such that this outer relaxation still provides a strict lower bound to the optimal solution of (11). Every instance where the solution of (11) was not optimal for (1) is marked by a "*".

The algorithms were coded in MATLAB R2016a, with calls to CPLEX 12.6 to solve all MILP and LP problems. Experiments were run on a Linux Cluster with five 20-core 2.4GHz Intel Xeon processors and 4 x 256GB RAM. For a fair comparison, all experiments were run with a single thread.

We compare the number of main iterations (number of times some node $[I^w, I^y]$ is selected) and total CPU time required by the different variants of the logical Benders method. It is important to point out that CPU time must not be taken too seriously, since it is not clear if hot-starts, in the MATLAB/CPLEX interface, works as efficiently as it could. Regardless of it, we set a time limit of 3,600 seconds not considering the time it takes to solve problem (11). Furthermore, we are not reporting the CPU time it takes to solve the big-$M$ formulation (11) described in Section 2.2, since that time is the same for every method. Additionally, since no sparsification is executed if the obtained node is "fathomable", the total number of sparsification calls is also provided. We do not report the number of sparsification calls for variant (1) since it is called in every iteration. We consider the number of main iterations as a reflection of the quality of both the node selected and the cut generated, so it is our main metric to observe. Still, generating strong cuts and finding strong new pieces to explore comes at the expense of CPU time. We complement some tables with charts to give a visual representation of the trade-offs between runtime and number of iterations. We analyze the experiments from smallest to largest. Unless otherwise noted, the instances include coupling constraints.

The first two instances (sizes $n_c = 100$ and $n_c = 300$), had a big-$M$ parameter set to 100. We can extract two main observations: (1) there is a significant decrease on the number of main iterations when switching from the SAT solver piece selection to the Tree guided node selection. This is illustrated by the gray and yellow lines from the graphs on figure 4, where the latter lies strictly below the former in all but 1 instance for $n_c = 100$. (2) $\ell_1$ and hybrid sparsifications dominate Splitting in all instances.

At a first glance, switching the piece selection method should not make any difference, but it actually does. The main advantage the tree-guided method has over SAT is that it checks whether the recently found open node can be fathomed immediately. In that case, no piece is selected and a cut is immediately generated, removing every single leaf under this node from subsequent piece selections. In the SAT selection case, a piece must be chosen every time, and the sparsification method (any sparsification method whatsoever) might fail to identify the same cut, and could therefore not remove all leaves as in the tree-guided case. This way, the overall algorithm will require more iterations to explore the full set of complementarity pieces. We can observe that the number of sparsification calls as compared to number of main iterations is below 50% on all Tree-guided variants.

One of the characteristics of the instances where Splitting was ineffective is that the starting incumbent was not the global optimum (Instances 8 and 10 for $n_c = 100$ and instances 4 and 6 for $n_c = 300$). An intuitive reason is that Hybrid and $\ell_1$ are more capable of dealing with non-optimal incumbents, since it treats each complementarity component individually. Even more, if we see $\ell_1$ as an effective surrogate for the $\ell_0$-norm formulation, both Hybrid and $\ell_1$ provide the sparsest possible cut, for the corresponding incumbent. Splitting, on the other hand, since it takes a more aggressive stance by rejecting a split if the value of the relaxation lies below the incumbent (so the higher the incumbent, the more likely the rejection is), it could therefore fail to find a cut with a sparsity level that lies between the current and the one obtained by accepting the split. We also see that Hybrid does not seem to provide any extra benefit compared to $\ell_1$ in terms of iterations as they differ by at most one. This could mean that $\ell_1$ already found a sparsest cut and, hence, calling the path-based procedure was unnecessary. We verified the generated cuts on each iteration and saw that the difference on sparsity level between Hybrid and $\ell_1$ was no larger than 2. With respect to CPU times, on the instances where all methods perform on par (in terms of iterations), Splitting is always the fastest among all three. This is expected, since in the "worst" case it only requires $O(\log(n_c))$ LP solves, per iteration. [2]

Table 1: Metrics for $n_c = 100$, $M = 100$

| Instance | Main iterations (Sparsification Calls) | | | | CPU times (s) | | | |
|---|---|---|---|---|---|---|---|---|
| | Base | Spl | $\ell_1$ | Hyb | Base | Spl | $\ell_1$ | Hyb |
| 2 | 27 | 10(6) | 8(4) | 8(4) | 2.3 | 2.02 | 0.61 | 0.62 |
| 3* | 40 | 6(2) | 6(2) | 5(2) | 1.17 | 0.52 | 0.43 | 0.47 |
| 4* | 33 | 2(1) | 2(1) | 2(1) | 2.54 | 0.28 | 0.33 | 0.28 |
| 5* | 25 | 16(7) | 8(5) | 8(5) | 1.07 | 0.17 | 0.17 | 0.13 |
| 6 | 28 | 6(2) | 6(2) | 5(2) | 1.53 | 0.74 | 0.55 | 0.55 |
| 1 | 37 | 4(1) | 4(1) | 4(1) | 1.54 | 0.38 | 0.34 | 0.29 |
| 7 | 22 | 7(2) | 7(2) | 7(2) | 2.33 | 0.33 | 0.23 | 0.28 |
| 8 | 46 | 22(4) | 11(3) | 11(3) | 0.94 | 0.4 | 0.37 | 0.36 |
| 9* | 44 | 6(4) | 6(4) | 6(4) | 4.19 | 0.98 | 0.51 | 0.57 |
| 10* | 34 | 42(6) | 12(4) | 12(4) | 2.59 | 0.33 | 0.39 | 0.39 |
| Geom. Mean | 32.7 | 8.42(2.82) | 6.32(2.45) | 6.1(2.45) | 1.83 | 0.48 | 0.37 | 0.36 |

Moving to size $n_c = 500$, with a big-$M$ value set to 1,000, we compare the methods on instances with and without coupling constraints, displayed on tables 3 and 4, respectively. We can observe an interesting phenomenon: All Tree-guided variants decrease both in iterations and CPU times, where as the Base method seems to do the exact opposite. We also see the same pattern as before, on the instances where the MILP did not find the optimal solution Splitting showed to be quite ineffective.

In the last set of instances, with $n_c = 1,000$, the MILP managed to find the optimal solution on all tested instances. We can see that in all, except instance 9,

[2] We write "worst" within quotes because if this case happens, the cut would have sparsity at most 2, i.e., very sparse.

Figure 4: Main iterations and CPU times: $n_c = 100$

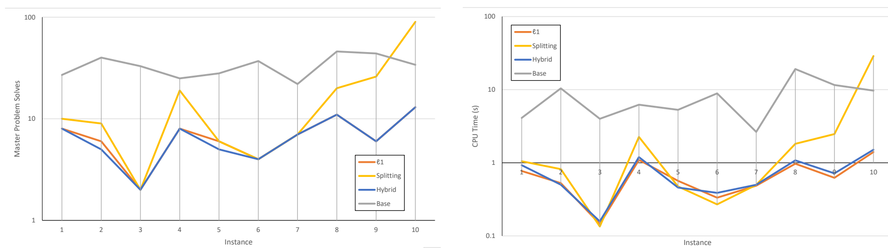


Table 2: Metrics for $n_c = 300$, $M = 100$

| Instance | Main iterations (sparsification calls) | | | | CPU times (s) | | | |
|---|---|---|---|---|---|---|---|---|
| | Base | Spl | $\ell_1$ | Hyb | Base | Spl | $\ell_1$ | Hyb |
| 1 | 71 | 11(5) | 8(4) | 8(4) | 27.18 | 1.36 | 1.56 | 1.76 |
| 2* | 75 | 33(9) | 15(5) | 14(5) | 38.74 | 3.56 | 2.66 | 3.05 |
| 3 | 87 | 11(7) | 10(7) | 9(6) | 40.82 | 1.87 | 2.63 | 2.75 |
| 4* | 108 | 86(9) | 32(10) | 32(10) | 67.6 | 12.78 | 4.92 | 5.7 |
| 5 | 126 | 5(3) | 5(3) | 5(3) | 62.19 | 0.81 | 1.12 | 1.13 |
| 6* | 89 | 106(19) | 31(11) | 31(11) | 53.65 | 13.32 | 4.92 | 5.94 |
| 7 | 106 | 5(3) | 5(3) | 5(3) | 34.65 | 0.8 | 1.04 | 1.11 |
| 8 | 136 | 6(3) | 6(3) | 6(3) | 87.94 | 0.88 | 1.13 | 1.23 |
| 9 | 94 | 12(5) | 11(4) | 11(4) | 46.04 | 1.32 | 1.8 | 1.87 |
| 10* | 85 | 13(6) | 9(5) | 9(5) | 44.17 | 1.76 | 1.88 | 2.16 |
| Geom. Mean | 95.73 | 15.6(5.81) | 10.59(4.92) | 10.4(4.85) | 47.63 | 2.14 | 2.03 | 2.23 |

the number of iterations was pretty stable among the three tree-guided variants. This situation benefits the splitting method since it dominates all instances in terms of CPU time. Similar to most of the instances from $n_c = 100$ going up, Hybrid does not seem to provide any benefit over $\ell_1$, and both still have the least number of iterations. The Base method failed to certify optimality within the 3,600 second limit on all instances. Its reporting was therefore withdrawn from the table.

# 5 Conclusions and Future Research

We introduced a new interpretation to the logical Benders approach for solving LPCCs from the perspective of branch-and-bound methods and exploited this relationship to provide a new way to select pieces and generate cuts. Numerical results showed that this method is more robust in the sense of keeping a consistent number of iterations along instances of the same size, even when the optimal solution is not provided, but at the expense of more time spent in the sparsification process. We also showed that the tree-guided approach for the piece selection outperforms the black-box SAT solver, which indicates that piece selection is indeed an important driver for the performance of the logical

Table 3: Metrics for $n_c = 500$, $M = 1,000$.

| Instance | Main iterations (Sparsification calls) | | | | CPU times (s) | | | |
|---|---|---|---|---|---|---|---|---|
| | Base | Spl | $\ell_1$ | Hyb | Base | Spl | $\ell_1$ | Hyb |
| 1 | 161 | 2(2) | 2(2) | 2(2) | 275.21 | 1.63 | 3 | 2.87 |
| 2* | 96 | 107(7) | 31(12) | 31(12) | 217.04 | 31.61 | 21.2 | 25.91 |
| 3* | 155 | 17(8) | 14(7) | 14(7) | 326.9 | 8.04 | 12.63 | 14.67 |
| 4 | 126 | 4(3) | 4(3) | 4(3) | 296.14 | 3.11 | 4.85 | 5.9 |
| 5 | 138 | 9(3) | 9(3) | 9(3) | 270.18 | 2.81 | 5.18 | 5.99 |
| 6* | 132 | 87(7) | 17(7) | 17(7) | 270.32 | 20.07 | 11.22 | 13.28 |
| 7 | 126 | 3(3) | 3(3) | 3(3) | 257.07 | 2.4 | 4.73 | 5.12 |
| 8* | 130 | 101(19) | 26(14) | 26(14) | 275.12 | 30.75 | 23.22 | 27.74 |
| 9 | 186 | 4(2) | 4(2) | 4(2) | 303.26 | 1.87 | 3.12 | 3.03 |
| 10 | 164 | 10(4) | 8(4) | 8(4) | 313.12 | 4 | 6.84 | 7.71 |
| Geom. Mean | 139.28 | 13(4.47) | 8.17(4.52) | 8.17(4.52) | 278.79 | 5.76 | 7.46 | 8.38 |

Table 4: Metrics for $n_c = 500$, $M = 1,000$. No coupling constraints.

| Instance | Main iterations (Sparsification calls) | | | | CPU times (s) | | | |
|---|---|---|---|---|---|---|---|---|
| | Base | Spl | $\ell_1$ | Hyb | Base | Spl | $\ell_1$ | Hyb |
| 1 | 202 | 3(2) | 3(2) | 3(2) | 444.87 | 2.37 | 3.55 | 4.16 |
| 2 | 184 | 4(4) | 4(4) | 4(4) | 467.73 | 3.79 | 6.86 | 7.55 |
| 3 | 211 | 7(4) | 7(4) | 7(4) | 600.85 | 3.95 | 6.75 | 7.19 |
| 4 | 210 | 3(3) | 3(3) | 3(3) | 458.73 | 2.95 | 5.25 | 5.3 |
| 5 | 190 | 20(4) | 20(4) | 20(4) | 481.77 | 5.22 | 8.01 | 9.14 |
| 6 | 139 | 20(6) | 19(5) | 19(5) | 334.23 | 6.66 | 9.96 | 10.95 |
| 7 | 124 | 7(4) | 5(3) | 6(4) | 324.33 | 4.74 | 5.76 | 8.04 |
| 8* | 180 | 20(6) | 10(5) | 9(5) | 452.65 | 7.67 | 9.77 | 10.56 |
| 9 | 205 | 2(2) | 2(2) | 2(2) | 434.34 | 1.53 | 2.62 | 2.87 |
| 10 | 125 | 5(3) | 5(3) | 5(3) | 262.49 | 3.01 | 4.86 | 5.06 |
| Geom. Mean | 173.61 | 6.53(3.57) | 5.86(3.34) | 5.91(3.44) | 415.83 | 3.79 | 5.88 | 6.56 |

Table 5: Metrics for $n_c = 1,000$, $M = 1,000$. No coupling constraints

| Instance | Main iterations (Sparsification calls) | | | CPU times (s) | | |
|---|---|---|---|---|---|---|
| | Spl | $\ell_1$ | Hyb | Spl | $\ell_1$ | Hyb |
| 1 | 5(3) | 5(3) | 5(3) | 24.29 | 51.06 | 58.9 |
| 2 | 5(3) | 5(3) | 5(3) | 30.88 | 52.3 | 63.28 |
| 3 | 5(3) | 5(3) | 5(3) | 24.15 | 51.95 | 49.52 |
| 4 | 7(3) | 7(3) | 7(3) | 33.48 | 61.97 | 60.21 |
| 5 | 5(2) | 5(2) | 5(2) | 23.4 | 40.41 | 50.07 |
| 6 | 6(4) | 6(4) | 6(4) | 35.73 | 70.38 | 68.46 |
| 7 | 3(1) | 3(1) | 3(1) | 10.59 | 21.48 | 25.14 |
| 8 | 5(2) | 6(2) | 5(2) | 24.58 | 36.16 | 39.97 |
| 9 | 16(6) | 8(3) | 8(3) | 48.87 | 61.44 | 65.05 |
| 10 | 7(3) | 7(3) | 7(3) | 28.21 | 53.2 | 61.99 |
| Geom. Mean | 5.69(2.71) | 5.38(2.51) | 5.27(2.51) | 26.5 | 47.22 | 51.37 |

Benders procedure. Future research may consider the extension of this branch-and-bound framework to Binary Constrained Quadratic Programs with Linear Complementarity Constraints (BCQPCCs).

## References


Achterberg T (2007) Constraint integer programming. PhD thesis, Technische University Berlin

Alvarez AM, Louveaux Q, Wehenkel L (2017) A machine learning-based approximation of strong branching. INFORMS Journal on Computing 29(1):185–195

Bai L, Mitchell JE, Pang JS (2013) On convex quadratic programs with linear complementarity constraints. Computational Optimization and Applications 54(3):517–554

Bard JF, Moore JT (1990) A branch and bound algorithm for the bilevel programming problem. SIAM Journal on Scientific and Statistical Computing 11(2):281–292

Belotti P, Bonami P, Fischetti M, Lodi A, Monaci M, Nogales-Gomez A, Salvagnin D (2016) On handling indicator constraints in mixed integer programming. Computational Optimization and Applications 65(3):545–566

Bonami P, Lodi A, Tramontani A, Wiese S (2015) On mathematical programming with indicator constraints. Mathematical Programming 151:191–223

Burdakov OP, Kanzow C, Schwartz A (2016) Mathematical programs with cardinality constraints: Reformulation by complementarity-type conditions and a regularization method. SIAM Journal on Optimization 26(1):397–425

Candes EJ, Wakin MB, Boyd SP (2008) Enhancing sparsity by reweighted $\ell_1$ minimization. Journal of Fourier analysis and applications 14(5-6):877–905

Cplex II (2010) 12.2 user's manual. Book 122 User's Manual, Series 122 User's Manual

Feng M, Mitchell JE, Pang JS, Shen X, Wächter A (2018) Complementarity formulations of $\ell_0$-norm optimization problems. Pacific Journal of Optimization 14(2):273–305

Fischer T, Pfetsch ME (2018) Branch-and-cut for linear programs with overlapping SOS1 constraints. Mathematical Programming Computation 10(1):33–68

Fischetti M, Monaci M (2013) Backdoor branching. INFORMS Journal on Computing 25(4):693–700

Fischetti M, Monaci M (2014) Exploiting erraticism in search. Operations Research 62(1):114–122

Fletcher* R, Leyffer S (2004) Solving mathematical programs with complementarity constraints as nonlinear programs. Optimization Methods and Software 19(1):15–40

Gomes CP, Selman B, Kautz HA (1998) Boosting combinatorial search through randomization. In: Proceedings of AAAI/IAAI 1998, AAAI, p online

Gomes CP, Kautz H, Sabharwal A, Selman B (2008) Satisfiability solvers. In: van Harmelen F, Lifschitz V, Porter B (eds) Handbook of Knowledge Representation, Elsevier, pp 89–134

Hooker JN, Ottosson G (2003) Logic-based Benders decomposition. Mathematical Programming 96(1):33–60

Hu J, Mitchell JE, Pang JS, Bennett KP, Kunapuli G (2008) On the global solution of linear programs with linear complementarity constraints. SIAM Journal on Optimization 19(1):445–471

Hu J, Mitchell JE, Pang JS (2012a) An LPCC approach to nonconvex quadratic programs. Mathematical programming 133(1-2):243–277

Hu J, Mitchell JE, Pang JS, Yu B (2012b) On linear programs with linear complementarity constraints. Journal of Global Optimization 53(1):29–51

Huang J (2007) The effect of restarts on the efficiency of clause learning. In: Proceedings of IJCAI-07, pp 2318–2323

Ibaraki T (1971) Complementary programming. Operations Research 19(6):1523–1529

Ibaraki T (1973) The use of cuts in complementary programming. Operations Research 21(1):353–359

Jeroslow RG (1978) Cutting-planes for complementarity constraints. SIAM Journal on Control and Optimization 16(1):56–62

Karzan FK, Nemhauser GL, Savelsbergh MW (2009) Information-based branching schemes for binary linear mixed integer problems. Mathematical Programming Computation 1(4):249–293

Leyffer S, López-Calva G, Nocedal J (2006) Interior methods for mathematical programs with complementarity constraints. SIAM Journal on Optimization 17(1):52–77

Lodi A, Zarpellon G (2017) On learning and branching: a survey. TOP 25(2):207–236

Luo ZQ, Pang JS, Ralph D (1996) Mathematical programs with equilibrium constraints. Cambridge University Press

Optimization G (2014) Inc.,“gurobi optimizer reference manual,” 2015. URL: http://www gurobi com

Williams R, Gomes CP, Selman B (2003) Backdoors to typical case complexity. In: Proceedings of IJCAI-2003, pp 1173–1178

Yu B (2011) A branch and cut approach to linear programs with linear complementarity constraints. PhD thesis, Department of Decision Sciences and Engineering Systems, Rensselaer Polytechnic Institute, Troy, NY 12180 USA

Yu B, Mitchell JE, Pang JS (2018) Solving linear programs with complementarity constraints using branch-and-cut. Mathematical Programming Computation DOI 10.1007/s12532-018-0149-2, URL https://doi.org/10.1007/s12532-018-0149-2